A Systematic Method for Development of Reactive Real-Time Systems

Alan Grigg,
British Aerospace Dependable Computing Systems Centre,
Department of Computer Science,
University of York,
York, UK

Neil Henderson
British Aerospace Dependable Computing Systems Centre,
Department of Computer Science,
University of Newcastle Upon Tyne,
Newcastle Upon Tyne, UK

Abstract. In this paper, we identify the need for a more systematic method to support the specification and analysis of reactive real-time systems and propose such an approach based on *real-time transactions*. Firstly, a graphical notation is outlined that can be used to specify the timing properties of real-time transactions. Secondly, a rigorous formal notation is proposed to support the expression of transaction functionality. Finally, a timing analysis method is described that can be used to predict the timing behaviour of the system at any stage of refinement of the formal model. The use of the approach is illustrated via a case study. We suggest that the application of a more systematic approach to the development of real-time systems can provide major cost benefits when considered over the life-cycle of the system.

INTRODUCTION

This paper describes the current results of an ongoing British Aerospace funded research activity within the Dependable Computing Systems Centre (DCSC) at the Universities of Newcastle and York. The research is focussed on providing a more systematic method for the development of reactive[1] real-time systems, from the specification of functionality and timing properties through to an implementation, by continual refinement of system properties. In order to achieve this objective, a real-time transaction model has been developed, consisting of :

- A graphical notation that can be used to specify the timing properties of a system and a rigorous formal notation to express its functionality. We advocate the use of a rigorous formal notation for the specification of system functionality, although the model can be adapted to use a less rigorous notation if the user wishes.
- A timing analysis method that can be performed at any stage of refinement of the system timing properties to predict the overall timing behaviour of the system.

It is often the case that errors in a system's specification, due to the relative timing of computations, are not identified until they manifest themselves as bugs during the testing stage. A considerable amount of costly rework is then required, to correct the code, and modify the specification and design documentation to reflect the changes that have been made. The formal notations have been developed as a useful abstraction to enable the recording and analysis of functional and timing properties of the system in the early to mid stages of the development cycle. The use of these rigorous notations will help to identify more of the errors early in the development process, before the coding stage, to reduce the amount of such rework.

[1] A reactive system is one which is constantly interacting with its environment, and where the behavior of the system is influenced by its speed relative to changes in the environment.

Graphical notations are used to specify a partial ordering of actions within the system, but it is important that this specification is kept sufficiently abstract so that event orderings are specified only where they are essential - for example, to ensure that an input is read before a related computation is carried out. This ensures that the final real-time scheduling solution for the system is not unnecessarily constrained, and the potential for maximal concurrency is retained. It is left up to the run-time scheduling solution to control the actual allocation of hardware resources and appropriate techniques can be used from the real-time systems domain that are both flexible and predictable.

Timing analysis is performed by constructing an abstract (implementation-independent) model of the system, including the perceived target hardware. The abstract model is successively refined to capture the timing properties of the system, so that at each stage of development, the abstract model represents a set of timing obligations on the subsequent stages of development. The technique is known as *reservation-based timing analysis*, as originally published in (Grigg 1999). This model has since been extended and integrated with the formal RTT model described below to give a process for top-down specification and analysis of timing properties for real-time sytsems.

In this paper, we describe the approach and illustrate its use via a case study – an Attitude Guidance Autopilot system, which is a simplified version of a missile system.

WHY DEVELOP A NEW FORMAL APPROACH?

Existing formal specification languages which are capable of specifying the behaviour of reactive systems, such as Petri-nets (ISO 1997), process algebras (for example CSP (Roscoe 1997)) and temporal logics, such as RTL (Jahanian 1986, Jahanian 1988), usually only refer to the ordering of input and output events, not the relationship between those inputs and outputs. Our aim is to characterise the behaviour of reactive systems more fully and this is not possible if the event based reactivity is separated from the functionality of the system.

The approach of Statecharts (Harel 1987) also has severe limitations and does not seem appropriate for systems with internal concurrency for the following reasons :

- States are an inaccurate abstraction of many concurrent and distributed systems. Engineers tend to refer to the mode of a system rather than its state, and while the two concepts are similar there are fundamental differences. For example, particularly with distributed systems, it can take some time for an entire system to respond to a change of mode. This can result in different parts of the same system being in different modes at the same time.
- The *synchrony hypothesis* inherent in the approach means that a statechart must finish responding to one set of inputs before the next set arrives. This requires systems to become “blind” to changes in their inputs once they have started a computation. While this may be true for small systems (although even then only usually only for sequential systems) at a higher level of abstraction a specification may be referring to a complete system, or sub system, and it is unlikely that this property will hold.
- Concurrent and distributed systems which are constructed to obey the synchrony hypothesis tend to be both inefficient and inflexible. Inefficiency arises because they require a fixed schedule that is constructed around the assumption of consistent worst-case behaviour - for example, the period of all the computations is driven by the algorithm that takes longest to execute. In turn, this is not flexible enough to support more reactive behaviour at run-time. Moreover, the resultant lack of flexibility can be a major contributor to development-time and in-service upgrade costs associated with the system.

SPECIFICATION OF REAL-TIME TRANSACTIONS

The real-time transaction (RTT) model (Haveman 1997) has been developed to provide a method to formally specify and reason about the functionality and timing of reactive distributed real-time systems using a new formal notation known as the transaction description language (TDL). The computational model has been kept sufficiently abstract to be applicable to systems that contain internal concurrency or distributed parallelism and so that it can be :

- reasoned about without reference to the underlying implementation;
- adapted to be used with different design methodologies and processes.

A RTT is a computation that is triggered either periodically by a clock or sporadically by one of its inputs, which then reads all of its inputs and produces a (potentially empty or complete) subset of its outputs. The functionality and relative timing of the inputs and outputs are specified using a graphical transaction state machine (TSM) notation.

We will first illustrate the AGA system using UML 'use cases' and then show the specification of part of the system using the RTT notations. Later, we will show how these notations can be used to support a timing analysis of the system.

The AGA system receives data from sensors, which it uses to calculate the attitude of the missile. This information is then passed to an Information Processing System (IPS), and is used together with information about the target, such as infra red image to provide an aim point for the missile which is input back to the AGA system. This aim point is then used together with the attitude of the missile to calculate guidance information for the missile. A UML 'use case' description of the AGA system is shown in Figure 1. This identifies two RTTs, the Attitude Guidance and the Autopilot.

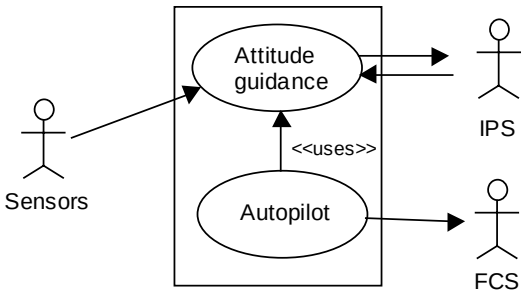

Figure 1. The AGA System

Figure 2 illustrates the TSM graphical notation applied to the AGA Autopilot RTT.

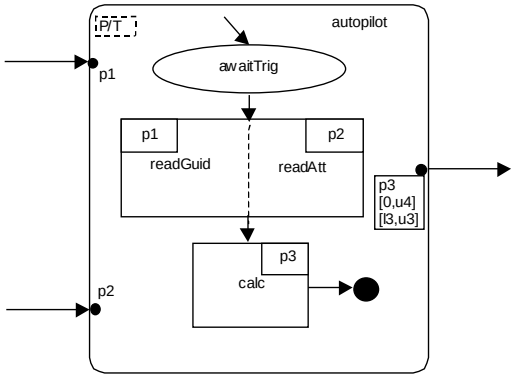

Figure 2. The AGA Autopilot RTT State Machine

The TSM notation is comprised of the following objects :

- Ports, where inputs are received from, and outputs are made to, either other RTTs or the environment.
- Static states, which represent potential blocking/descheduling points such as where it is necessary to wait for data to become available in a buffer.
- Dynamic states, each comprising a single computation that reads its inputs once and writes its outputs on completion.
- Hierarchical states, which themselves contain a RTT, and therefore make it possible to refine the specification of a RTT in a recursive hierarchical manner.
- Composite states, which are comprised of a number of dynamic states, or hierarchical states, which must be non interfering and can be executed in any order, or concurrently. Non interference means that the operations in each state within a composite state must read from and write to

different ports and use different sub-sets of the local state of a RTT.

- Transitions, which lead from one state to another.
- Termination states, which a RTT enters when it has finished its computation. That instance of the RTT then ceases to exist.

We now illustrate this terminology with reference to Figure 2. The RTT is represented by a rectangle with rounded corners, the ports by solid spots on its perimeter, and dataflows by arrows to or from the respective ports. The initial state is indicated by a transition with no source. Timing constraints are shown alongside the port in the TSM, for instance there may be upper and lower bounds for data to be read from or written to a port. For example, the output at port p8 must take place within l3 and u3 after the RTT is triggered and there is a deadline of u4 on the calculation of the output. There is a lower bound of 0 shown with the deadline and it is possible to show a non zero lower bound on the completion of the calculation.

A new instance of the autopilot RTT is created on receipt of the trigger every T time units (shown in the diagram in the dashed box at the top left of the TSM – where P stands for periodic). A RTT may be triggered sporadically by the arrival of data at one of its inputs in which case the minimum separation time of the triggers is shown. It then reads the inputs at ports p1 and p2 (these actions can be executed concurrently) and stores them as part of its local state, and enters state ‘calc’ where the data to be output is calculated and output at port p3. Finally it enters its termination state and ceases to exist. Each of the dynamic states in the RTT has a VDM (ISO 1993) operation associated with it, with pre and post conditions used to express the computation that is carried out in the state.

All of the information about a RTT is recorded in a tabular format which contains the following information :

- The names and brief descriptions of inputs, outputs and local state variables, together with details of any timing constraints, such as deadlines for outputs or windows during which inputs must be read and outputs written.
- Functionality - this section would contain the TSM description of the RTT, details of any types imported from other RTTs, a description of the operations carried out in the dynamic states of the RTT and any auxillary functions called.
- Guard - a RTT can have a guard which is a boolean condition over its inputs and local state variables and is evaluated when it is triggered. The RTT only executes if the guard evaluates to true.
- Invariants, condition(s) evaluated over the local state of a RTT which must hold when it is executed.

Information is extracted from this table to support analysis under the reservation-based timing analysis model, as described below.

TIMING ANALYSIS OF REAL-TIME TRANSACTIONS

Since the original development of the reservation-based timing analysis (RBA) model and its publication in (Grigg 1999), the model has been extended and applied alongside the RTT formal method described above. Below, the latest version of the RBA model is summarised in the context of the RTT formal method and the AGA case study.

The RBA model supports the notion of continual refinement of system timing properties through the provision of *nested* transactions. At any level of nesting, a transaction is described in terms of *activities* which capture its detailed timing and resource usage requirements. Activities are analogous to the dynamic states referred to above in the description of the formal RTT model. Activities can themselves be refined (alongside the formal model) to produce further levels of nesting in the RBA model. Hence, at any stage of refinement, a ‘precedence graph’ of activities or dynamic states can be extracted from the TSM and the RBA timing analysis applied.

The following additional information is associated with each activity or dynamic state to describe the resource usage characteristics of a transaction :

- A reservation budget, *V*, to define the *rate* at which resource is consumed; *V* is a real number in the interval (0,1].
- A best-case and worst-case total resource requirement, [*c*, *C*].

The RBA model can then determine the local delays (and delay variations, *ie.* jitter) associated with each dynamic state and consequently the end-to-end timing behaviour for the RTT in question. RBA incorporates an abstract (technology-independent) model of the run-time operation of each physical resource in the system, typically processing and communication resources. Run-time scheduling policies must, in turn, reflect the assumptions made in the abstract model for the off-line timing analysis results to remain applicable. The RBA method is applied to the case study below.

Figure 3 illustrates the precedence graph of dynamic states for the AGA Autopilot RTT.

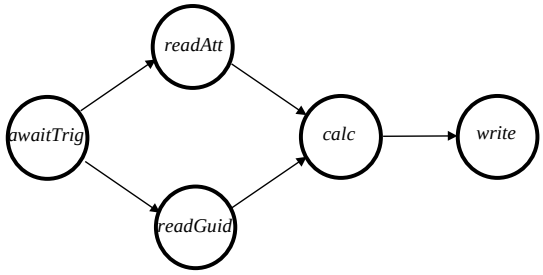

Figure 3. The AGA Autopilot RTT Precedence Graph

Table 1gives some example local delays calculated by the RBA model from a given set of resource usage requirements and reservation budgets. The minimum and maximum predicted delays are denoted by r and R, respectively. These local delays then form the input to an overall end-to-end delay calculation.

	$c_{i,j}$	$C_{i,j}$	$V_{i,j}$	$r_{i,j}$	$R_{i,j}$
awaitTrig	1	3	0.1	10	30
readAtt	1	2	0.2	5	10
readGuide	2	3	0.2	10	15
calc	3	8	0.4	8	20
write	2	3	0.2	10	15

Table 1. The AGA Autopilot RTT Timing Analysis (Local Delays)

The end-to-end delay calculation may be performed manually to give a best-case of 38 and a worst-case of 80. More recently, however, the RBA method can be applied more conveniently with tool assistance. This has been enabled by a MATLAB/Simulink formulation of the RBA model. This tool has been developed to allow engineers to rapidly construct and analyse end-to-end transactions under RBA conditions. The tool allows the engineer to apply the RBA method without detailed knowledge of the underlying analytial model. The RBA tool has been used to construct and analyse the example AGA Autopilot transaction. Figure 4 shows a plot of the predicted end-to-end response time for one hundred simulated executions of the example transaction. During simulation, the execution time of each activity is allowed to vary randomly between user-defined limits, [*c*, *C*].

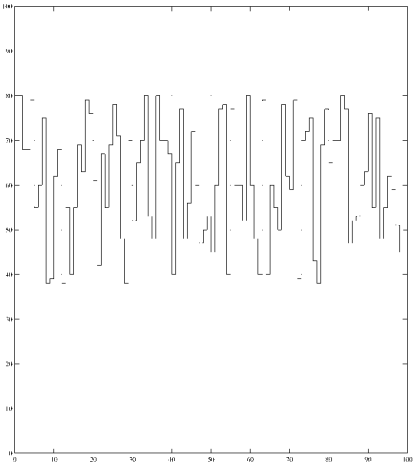

Figure 4. The AGA Autopilot RTT Timing Analysis (End-to-End Delays)

SUMMARY

The paper has described a systematic method for specification and analysis of reactive, real-time systems. We believe that the application of our approach to the development of reactive real-time systems can provide major cost benefits when considered over the life-cycle of the system.

REFERENCES

(Grigg 1999) A Grigg,, NC Audsley, MA Fletcher, AS Wake, “A Method for Design and Analysis of Next Generation Aircraft Computer Systems”, Proc. of INCOSE Conference on Systems Engineering, 1999.

(Harel 1987) D Harel, “Statecharts: A visual formalism for complex systems”, Science of Computer Programming”, 1987.

(Haveman 1997) J Haveman, S E Paynter, J M Armstrong, “A transaction model for real-time systems”, Technical Report No 607, University of Newcastle, 1997.

(ISO 1993) The ISO/VDM-SL Committee, “VDM Specification Language”, 1993 (Draft).

(ISO 1997) ISO/IEC JTC 1 on Information Technology, “Petri Nets Standard – Project 7.19.3 (Petri Nets), Version 3.4 (Draft)”, 1997.

(Jahanian 1986) F Jahanian, A K Mok, “Safety analysis of timing properties in real-time systems”, IEEE Transactions on Software Engineering, SE-12(9):890-904, 1986.

(Jahanian 1988) F Jahanian, A K Mok, D A Stuart, “Formal specification of real-time systems”, Technical Report TR-88-25, The University of Texas at Austin, 1988.

(Roscoe 1997) A W Roscoe, “The Theory and Practice of Concurrency”, ISBN 0-13-674409-5, Prentice-Hall, 1997.

BIOGRAPHY

Alan Grigg received an honours degree in Mathematics at Thames Polytechnic (now University of Greenwich) in 1985. He subsequently joined British Aerospace; eventually working on the specification and design of a software architecture for military IMA systems. He was a British Aerospace representative for the multi-national, military IMA programme ASAAC. In 1996, he joined the British Aerospace Dependable Computing Systems Centre at the University of York to pursue research into real-time system design and timing analysis.

Neil Henderson worked in a leading British Clearing Bank from 1975 to 1994. He subsequently obtained an honours degree in Computing Science at the University of Newcastle upon Tyne in 1998, after which he joined the British Aerospace Dependable Computing Systems Centre at the University of Newcastle upon Tyne to pursue research into the formal specification of safety critical reactive real-time distributed computing systems.